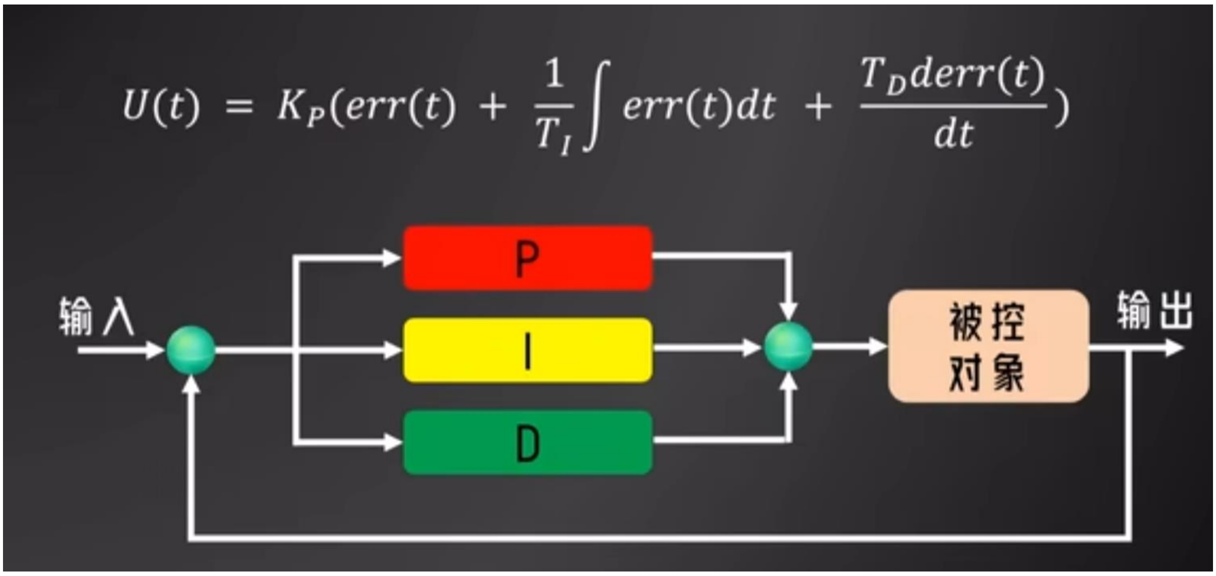
个人理解 PID，并编程实践

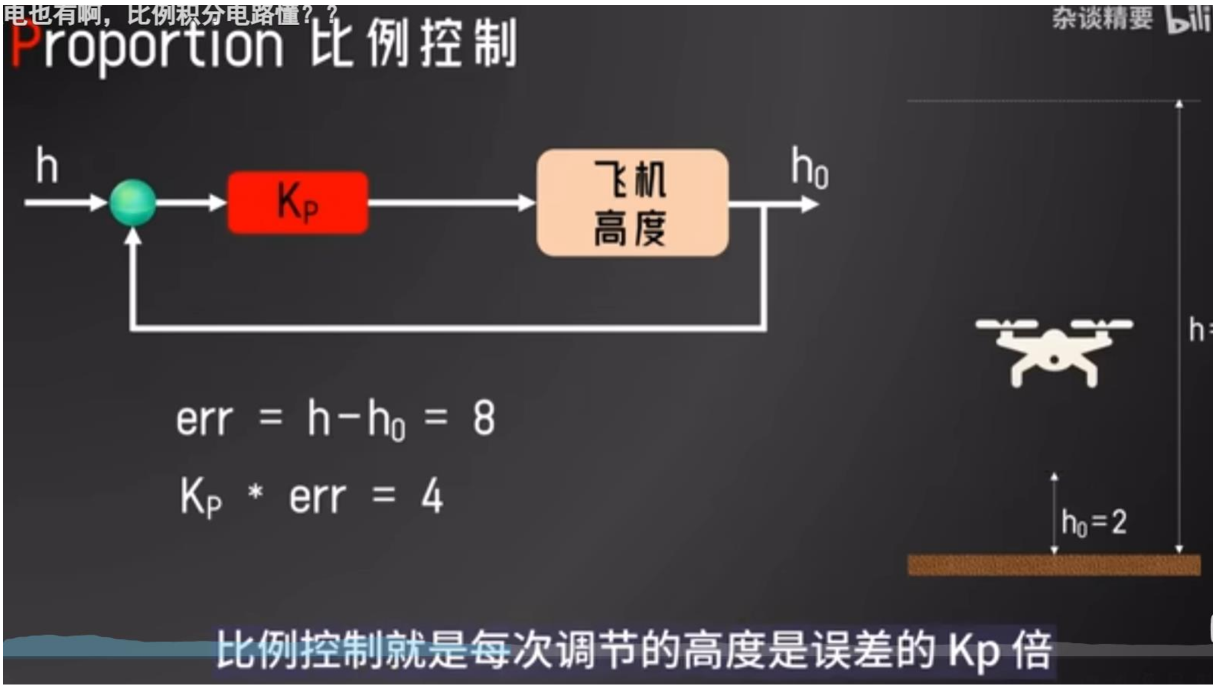
参考视频 https://www.bilibili.com/video/BV1GD4y1x7bV/

首先明确 error=预期值-测量值

Kp,Ki,Kd 均>=0



1. 比例控制 proportion =Kp*error(t)

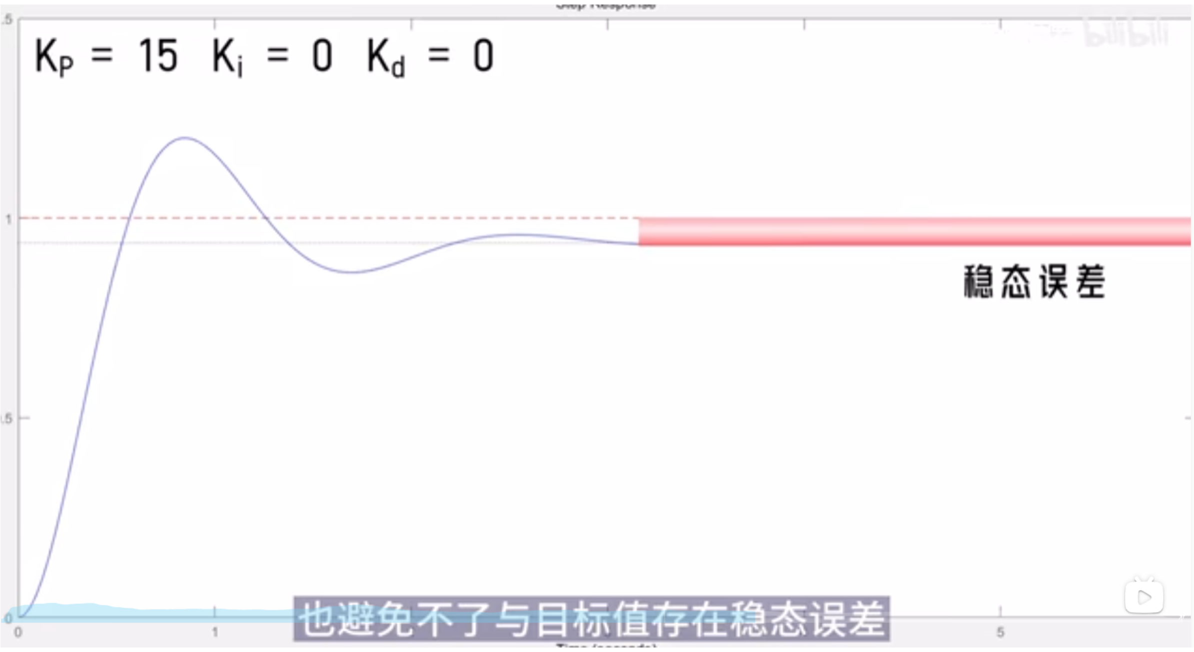


可知 KP 越大，系统反应越快，也就是与目标差快速缩小

受到实际环境的干扰，例如风力，会将无人机往下吹，就有可能抵消 Kp*error 的努力（我们的

控制系统还是傻傻的 Kp*error，而 error 持续不变)，此时后续若干个环节将持续 KP*err。但无人机永远无法上升到指定高度。这就是【稳态误差】*稳态误差*是系统从一个稳态过渡到新的稳态，或系统受扰动作用又重新平衡后，系统出现的偏差。

KP 自己无法解决稳态误差问题！



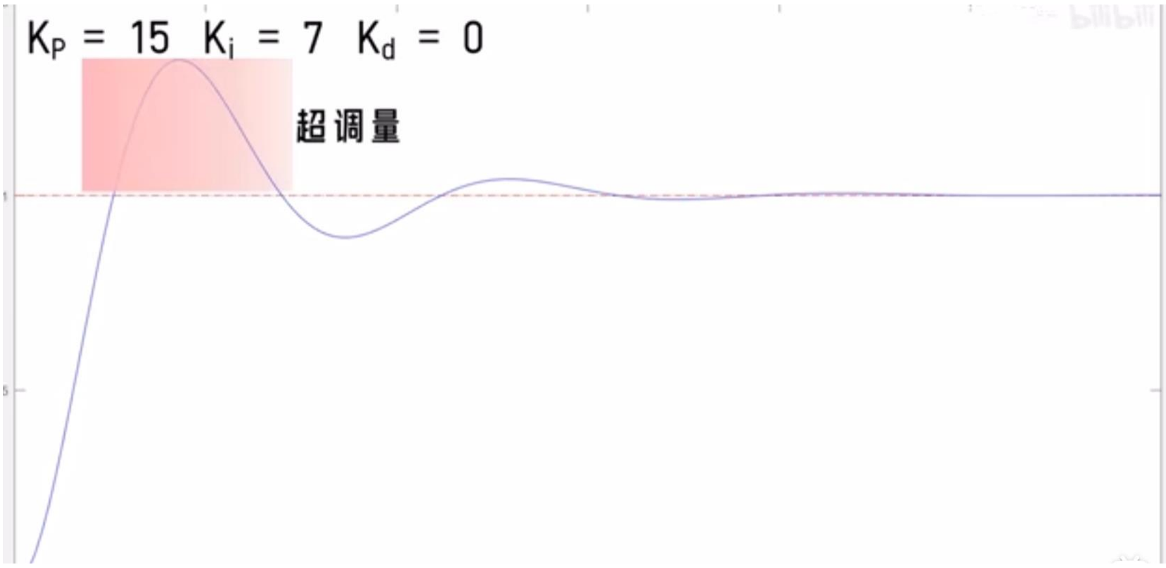
2. 积分控制 Integration

=Ki[Σerror(0)+error(1)+error(2)+....+error(t)]

就是为了消除稳态误差！按照我的理解，应当乘以 dt 再参与运算，但是编程后发现如果不改变 ki 的话效果很差。【由于 dt=1/sampletime 是个常数，因此实际效果等价于 ki 被缩放了，所以不要*dt 也罢】

对过去所有的误差 error 求和，离散状态下就是对所有的 error 求和喽！

但系统仍然不完美，会出现超调量过大的问题



3. 差分控制 Differential

=Kd*[errot(t)-errot(t-1)]=Kd*Δerror

=Kd*[setpoint-input(t)]-[setpoint-input(t-1)]

=Kd*[-input(t)+input(t-1)]

=-Kd*[Δinput]

按照我对公式的理解应当再除以一个 dt，编程表明效果确实不好。
【由于 sampletime=1/t 是个固定值，实际上会让 kd 按照比例变大或缩小，如果这样理解的话，单纯调节 kd 就好，不/dt 也 ok】

为了抑制超调量，施加反作用力

分析可知当

- 测量值<预期值,因为 error=预期值-测量值，理想情况下(测量值上升)，error 不断变小且>0，此时 Δerror<0。因为 error>0，P+I 也>0，而此时微分部分 D<0 是负值，D 在施加反作用，减少超调量。
- 某个时刻 t,error(t)=0。之后如果发生了超调
- 测量值>预期值,因为 error=预期值-测量值，理想情况下（测量值下降），此后 error 不断变大且<0，此时 Δerror>0。因为 error<0，P+I 也<0，而此时微分部分 D>0 为正值，D 在施加反作用，减少超调量。

【二】编码实践

基于上述理解，自行编写一个 PID 控制直流电机的学习案例,效果还不错

```
/*AB 相编码器直流电机=uno 引脚*/
#define MORTOR_C1 2  //中断口 0 是 2 INT0
#define MORTOR_C2 12  // Right motor
// #define MORTOR_C1 3            //中断口 1 是 3 INT1
// #define MORTOR_C2 13            //Left motor
#define MORTOR_REDUCTION 90       //减速比
#define MORTOR_PULSE_PER_ROUND 11  //编码器每周输出脉冲个数
/*
L298N 模块连接 =uno 引脚
*/
#define L298N_EN_MORTOR 10  //B-Right motor。PWM 默认频率 490hz
#define L298N_MORTOR_IN_X 6
#define L298N_MORTOR_IN_Y 5

// #define L298N_EN_MORTOR 9  ///A-Left motor。PWM 默认频率 490hz
// #define L298N_MORTOR_IN_X 8
// #define L298N_MORTOR_IN_Y 7
/*PID 相关参数*/
unsigned int sample_time = 100;  //100ms 为默认采样时间
unsigned int dt=0.1;//dt=0.1s
unsigned long start_time = millis();
volatile double pulse = 0;  //如果是正转，那么每计数一次自增 1，如果是反转，那么每计数一次自减 1
double kp = 1.5, ki = 0.5, kd = 0.01;

double target_wheel_vel =30;                    //目标转速
```

```
double measure_wheel_vel = 0, previous_measure_wheel_vel = 0;  //测量转速
double error = 0, previous_error = 0, sum_error = 0;          //转速误差
double out_pwm = 0, previous_out_pwm = 0;                    //pid 计算得到的 pwm
void pulse_C1() {
  //2 倍频计数实现
  //手动旋转电机一圈，输出结果为 一圈脉冲数 * 减速比 * 2
  if (digitalRead(MORTOR_C1) == HIGH) {

    if (digitalRead(MORTOR_C2) == HIGH) {  //A 高 B 高
      pulse++;
    } else {  //A 高 B 低
      pulse--;
    }

  } else {

    if (digitalRead(MORTOR_C2) == LOW) {  //A 低 B 低
      pulse++;
    } else {  //A 低 B 高
      pulse--;
    }
  }
}

//速度更新函数
void getWheelVel() {
  //获取当前速度
  long right_now = millis();
  long past_time = right_now - start_time;  //计算逝去的时间
  if (past_time >= sample_time) {          //如果逝去时间大于等于一个计算周期
```

```
  noInterrupts();                                   // 关闭所有中断
  previous_measure_wheel_vel = measure_wheel_vel;                       //保留轮速，在计算新的
  measure_wheel_vel = (double)pulse / (2 * MORTOR_PULSE_PER_ROUND * MORTOR_REDUCTION) / past_time * 1000 * 60;  //车轮每分钟转数
  //4.重置开始时间和脉冲
  start_time = right_now;
  pulse = 0;
  interrupts();  // 启动中断允许
        /*匹配串口绘图器的多参数格式*/
  // Serial.print("sum_error:");
  // Serial.print(sum_error);
  Serial.print(",measure_wheel_vel:");
  Serial.print(measure_wheel_vel);
  Serial.print(",target_wheel_vel:");
  Serial.println(target_wheel_vel);
  /*匹配串口绘图器的多参数格式*/

  //开始 pid
  previous_out_pwm = out_pwm;
  previous_error = error;  //保留上次的误差
  error = target_wheel_vel - measure_wheel_vel;

sum_error += error;  //求和
  out_pwm = kp * error + ki * (sum_error) + kd * (error - previous_error);  //计算 pid

  if (out_pwm > 255) {
    out_pwm = 255;
    // sum_error=0; 何时清零合适呢？还是一直带着
```

```
    }
    if (out_pwm < 0) {
      out_pwm = 0;
      // sum_error=0;
    }
  }
}
void setup() {
  // put your setup code here, to run once:
  // put your setup code here, to run once:
  Serial.begin(57600);  //设置波特率
  /*设置编码输入*/
  pinMode(MORTOR_C1, INPUT);
  pinMode(MORTOR_C2, INPUT);
  /*设置 L298N 的输出引脚*/
  pinMode(L298N_EN_MORTOR, OUTPUT);
  pinMode(L298N_MORTOR_IN_X, OUTPUT);
  pinMode(L298N_MORTOR_IN_Y, OUTPUT);

  /*0 号中断*/
  attachInterrupt(0, pulse_C1, CHANGE);  //当电平发生改变时触发中断 0 函数
  /*1 号中断*/
  // attachInterrupt(1, pulse_C1, CHANGE);   //当电平发生改变时触发中断 0 函数
  digitalWrite(L298N_MORTOR_IN_Y, HIGH);  //给高电平
  digitalWrite(L298N_MORTOR_IN_X, LOW);   //给低电平
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(100);
```

```
  getWheelVel();
  analogWrite(L298N_EN_MORTOR, out_pwm);
}
```

【三】使用已有的库

也可以在使用 arduino 控制直流编码电机过程中，使用了库

https://github.com/br3ttb/Arduino-PID-Library

fork 了一份进行阅读批注

https://gitee.com/zw-ncist/Arduino-PID-Library

```
/**********************************************************************************************
 * Arduino PID Library - Version 1.2.1
 * by Brett Beauregard <br3ttb@gmail.com> brettbeauregard.com
 *
 * This Library is licensed under the MIT License
 **********************************************************************************************/

#if ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

#include <PID_v1.h>
```

```
/*Constructor (...)*********************************************************
 *    The parameters specified here are those for for which we can't set up
 *    reliable defaults, so we need to have the user set them.
 ***************************************************************************/
PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int POn, int ControllerDirection)
{
    myOutput = Output;//系统输出值，比如 pwm
    myInput = Input; //系统输入值，比如电机或轮胎的转速
    mySetpoint = Setpoint; //这个是预计目标，例如期待电机和轮胎的转速，注意单位要一致
    inAuto = false;//表明当前是否处于自动模式

    PID::SetOutputLimits(0, 255);                    //default output limit corresponds to
      //系统的输出范围，arduino 默认就是 0-255
      //the arduino pwm limits

    SampleTime = 100;//采样时间                                        //default Controller Sample Time
is 0.1 seconds

    PID::SetControllerDirection(ControllerDirection);//设置控制方向，例如 pwm 增加，电机转速
增加，这就是 DIRECT
    PID::SetTunings(Kp, Ki, Kd, POn);//优化用户提供的系数

    lastTime = millis()-SampleTime;//最近一次采样周期的开始位置？
}

/*Constructor (...)*********************************************************
 *    To allow backwards compatability for v1.1, or for people that just want
 *    to use Proportional on Error without explicitly saying so
 ***************************************************************************/
```

```
PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int ControllerDirection)
    :PID::PID(Input, Output, Setpoint, Kp, Ki, Kd, P_ON_E, ControllerDirection)
{

}



/* Compute() **********************************************************************
 *     This, as they say, is where the magic happens.  this function should be called
 *   every time "void loop()" executes.  the function will decide for itself whether a new
 *   pid Output needs to be computed.  returns true when the output is computed,
 *   false when nothing has been done.
 *   注意这个方法，在 loop 中每次调用。该方法会决定是否一个新的输出需要被计算
 **********************************************************************************/
bool PID::Compute()
{
   if(!inAuto) return false;
   unsigned long now = millis();
   unsigned long timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /*Compute all the working error variables*/
      double input = *myInput; //控制系统输入，例如转速 v(t)
      double error = *mySetpoint - input;//输入与期望的误差， error(t)
      double dInput = (input - lastInput);//lastInput=上一次 compute 时的系统输入 v(t-1)
      outputSum+= (ki * error);//?和公式不一样，不应该历史误差之和么【积分项 I，与公式不
一致】
      //检查代码可知 outputSum 从不清零，也是累加和，但公式中是误差之和，不带系数
```

```
      /*Add Proportional on Measurement, if P_ON_M is specified*/// ? 测量加入比例
http://brettbeauregard.com/blog/2017/06/introducing-proportional-on-measurement/
      if(!pOnE) outputSum-= kp * dInput;  //使用 Proportional on Measurement【比例项 P，这
是 POM】

      if(outputSum > outMax) outputSum= outMax;
      else if(outputSum < outMin) outputSum= outMin;

      /*Add Proportional on Error, if P_ON_E is specified 给误差加比例系数，如果 P_ON_E 有效*/
         double output;//这是真正的输出，要复制给 myOutput 的！【保存最终结果】
      if(pOnE) output = kp * error;//传统的 Proportional on Error【比例项 P，和公式一致】
      else output = 0;

      /*Compute Rest of PID Output*/
      output += outputSum - kd * dInput;//差分做了推导，消去了 setpoint【差分项 D，和公式
一致】

          if(output > outMax) output = outMax;
      else if(output < outMin) output = outMin;
          *myOutput = output;

      /*Remember some variables for next time*/
      lastInput = input;
      lastTime = now;
          return true;
   }
   else return false;
}
```

```
/* SetTunings(...)*************************************************************
 * This function allows the controller's dynamic performance to be adjusted.
 * it's called automatically from the constructor, but tunings can also
 * be adjusted on the fly during normal operation
 * 动态调整控制器的动态性能
 ******************************************************************************/
void PID::SetTunings(double Kp, double Ki, double Kd, int POn)
{
   if (Kp<0 || Ki<0 || Kd<0) return;//小于零不行

   pOn = POn;
   pOnE = POn == P_ON_E;

   dispKp = Kp; dispKi = Ki; dispKd = Kd;

   double SampleTimeInSec = ((double)SampleTime)/1000;//用秒来表示的采样周期
   kp = Kp;
   ki = Ki * SampleTimeInSec;//为积分做准备
   kd = Kd / SampleTimeInSec;//为微分做准备

  if(controllerDirection ==REVERSE)
   {
      kp = (0 - kp);
      ki = (0 - ki);
      kd = (0 - kd);
   }
}

/* SetTunings(...)*************************************************************
 * Set Tunings using the last-rembered POn setting
```

```
 ******************************************************************************/
void PID::SetTunings(double Kp, double Ki, double Kd){
    SetTunings(Kp, Ki, Kd, pOn);
}

/* SetSampleTime(...) *********************************************************
 * sets the period, in Milliseconds, at which the calculation is performed
 ******************************************************************************/
void PID::SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime
                      / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

/* SetOutputLimits(...)****************************************************
 *     This function will be used far more often than SetInputLimits.  while
 *  the input to the controller will generally be in the 0-1023 range (which is
 *  the default already,)  the output will be a little different.  maybe they'll
 *  be doing a time window and will need 0-8000 or something.  or maybe they'll
 *  want to clamp it from 0-125.  who knows.  at any rate, that can all be done
 *  here.
 *  设置控制系统的输出限制，例如控制电机的 pwm
 *   控制系统的输入限制默认为 0-1023（SetInputLimits 没找到这个方法）
 **************************************************************************/
```

```
void PID::SetOutputLimits(double Min, double Max)
{
   if(Min >= Max) return;
   outMin = Min;
   outMax = Max;

   if(inAuto)
   {
          if(*myOutput > outMax) *myOutput = outMax;
          else if(*myOutput < outMin) *myOutput = outMin;

          if(outputSum > outMax) outputSum= outMax;
          else if(outputSum < outMin) outputSum= outMin;
   }
}

/* SetMode(...)****************************************************************
 * Allows the controller Mode to be set to manual (0) or Automatic (non-zero)
 * when the transition from manual to auto occurs, the controller is
 * automatically initialized
 * 设置模式，用来设置控制器的模式，为手动=0，还是自动=非 0
 * 当从手动切换到自动时，会初始化 PID
 ******************************************************************************/
void PID::SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
    if(newAuto && !inAuto)
    {  /*we just went from manual to auto*/
        PID::Initialize();
    }
```

```
    inAuto = newAuto;
}

/* Initialize()****************************************************************
 *    does all the things that need to happen to ensure a bumpless transfer
 *  from manual to automatic mode.
 *该方法为了无缝从手动切换到自动模式
 ******************************************************************************/
void PID::Initialize()
{
   outputSum = *myOutput;
   lastInput = *myInput;
   if(outputSum > outMax) outputSum = outMax;
   else if(outputSum < outMin) outputSum = outMin;
}

/* SetControllerDirection(...)*************************************************
 * The PID will either be connected to a DIRECT acting process (+Output leads
 * to +Input) or a REVERSE acting process(+Output leads to -Input.)  we need to
 * know which one, because otherwise we may increase the output when we should
 * be decreasing.  This is called from the constructor.
 * DIRECT 模式意味着  Output 增加导致 Input 增加
 * REVERSE 反之，Output 增加导致 Input 减少
 ******************************************************************************/
void PID::SetControllerDirection(int Direction)
{
   if(inAuto && Direction !=controllerDirection)
   {
	  kp = (0 - kp);
      ki = (0 - ki);
```

```
    kd = (0 - kd);
  }
  controllerDirection = Direction;
}

/* Status Funcions*************************************************************
 * Just because you set the Kp=-1 doesn't mean it actually happened.  these
 * functions query the internal state of the PID.  they're here for display
 * purposes.  this are the functions the PID Front-end uses for example
 ******************************************************************************/
double PID::GetKp(){ return  dispKp; }
double PID::GetKi(){ return  dispKi;}
double PID::GetKd(){ return  dispKd;}
int PID::GetMode(){ return  inAuto ? AUTOMATIC : MANUAL;}
int PID::GetDirection(){ return controllerDirection;}
```

期间涉及到了 PoM

阅读

http://brettbeauregard.com/blog/2017/06/introducing-proportional-on-measurement/

So What is Proportional on Measurement?

Similar to Derivative on Measurement, PonM changes what the proportional term is looking at. Instead of error, the P-Term is fed the current value of the PID input.

Proportional on Error:

$$\text{Output} = K_p e(t) + K_I \int e(t)dt - K_d \frac{d\text{Input}}{dt}$$

Proportional on Measurement:

$$\text{Output} = -K_p[\text{Input}(t) - \text{Input}_{init}] + K_I \int e(t)dt - K_d \frac{d\text{Input}}{dt}$$